ANNUAL REPORT
ON
AUTOMATED ASSEMBLY IN SPACE

BY

DR. SADANAND SRIVASTAVA
BOWIE STATE UNIVERSITY
BOWIE, MD 20715

FOR

MR. GARY K. JONES -(CODE 731.4)
APPLIED ENGG. DIVISION
GODDARD SPACE FLIGHT CENTER
GREENBELT, MD 20771.

UNDER THE NASA RESEARCH GRANT # NAG5-1054.

GUIDELINES AND RULES FOR AUTOMATED ASSEMBLY BY ROBOTS IN SPACE

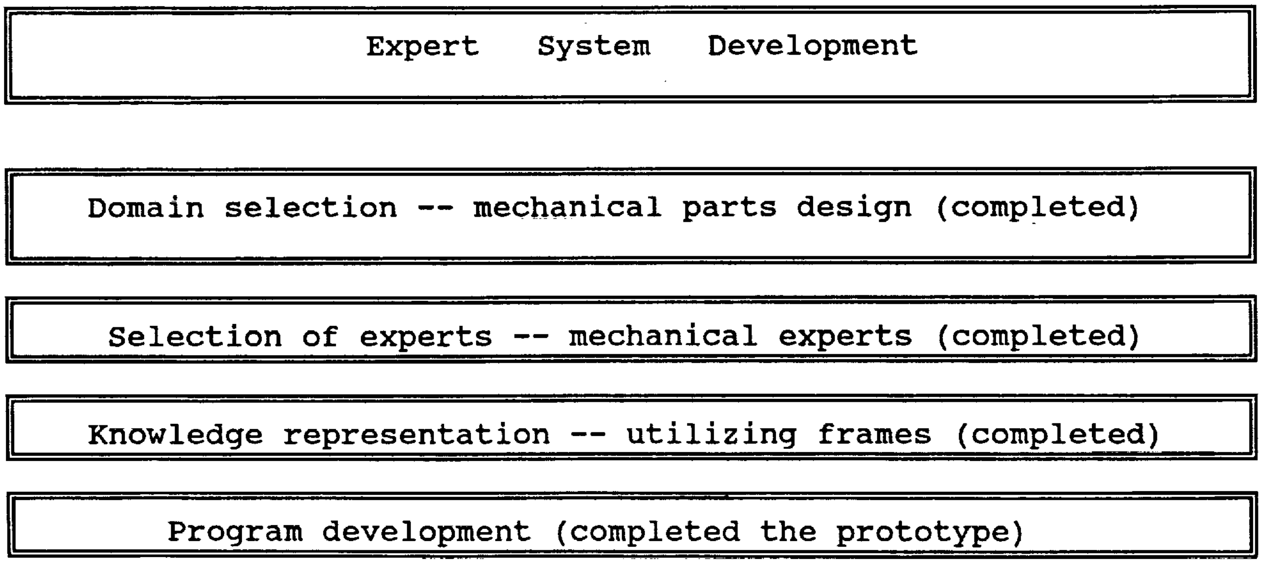
ABSTRACT

This report outlines the work of designing the Expert System for a "Mechanical design system". Two different implementation approaches have been described. One is coded in "C" and the other is realized by a software package --"Exsys". The first method has the advantage of greater flexibility and quicker responses, while the latter one is easier to develop. This report suggests the feasible ways to establish a real mechanical intelligent design system applying artificial intelligence techniques so that the products designed by this system could best meet the requirements for space assembly.

In phase one, the essential functions of the mechanical parts design system have been discussed and defined. The architecture of the system has been described and some design issues have also been discussed (such as explanation interface, knowledge representation, knowledge acquisition and search implementation). (See Appendix 1).

SECOND PHASE

Based on the results of the first phase, we developed a prototype to simulate the mechanical parts design. The following flow chart (Fig. 1) describes step by step work.



Expert System Development

Domain selection -- mechanical parts design (completed)

Selection of experts -- mechanical experts (completed)

Knowledge representation -- utilizing frames (completed)

Program development (completed the prototype)

Fig. 1

EXPERT SYSTEM DEVELOPMENT

Two different methods of development have been pursued. One of our approaches to develop an expert systems program is to find a software tool or shell, which is a software package that provides facilities to aid in expert system development. The second method is to develop the entire expert system by coding in "C".

1. The First Method: [Appendix 2]

The "Exsys" is one such software that helps us create an environment of the expert system with the features we need. "Exsys" is an expert system shell run under IBM PC/XT/AT. It has the following features:

A. List the steps needed to create a knowledge base.

B. Enter rules into an expert system.

C. Create a set of rules for a knowledge base.

D. Run an expert system program.

The advantage of this software is that it is easy to develop and needs very little coding. So even a non-computer-technical person can build an expert system by using this software.

It is to be remembered that a fully developed operational system will need not only an expert system but also an interface with intelligent CAD system together with a smart database (Fig 2).

2. The second method: [Appendix 3]

The second method is to build an expert system by "C" coding. The advantage of this approach is its flexibility. It can easily add a special feature to the system or modify the existing features according to the users' requirements. It also has a very good response time.

A: ARCHITECTURE

It' architecture is shown in fig. 2.

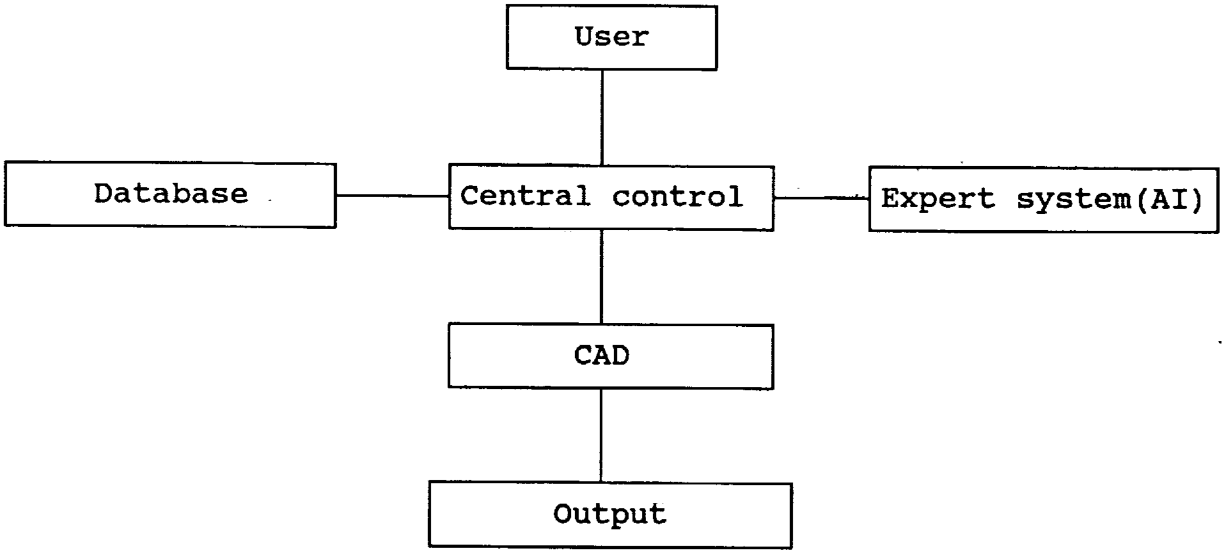
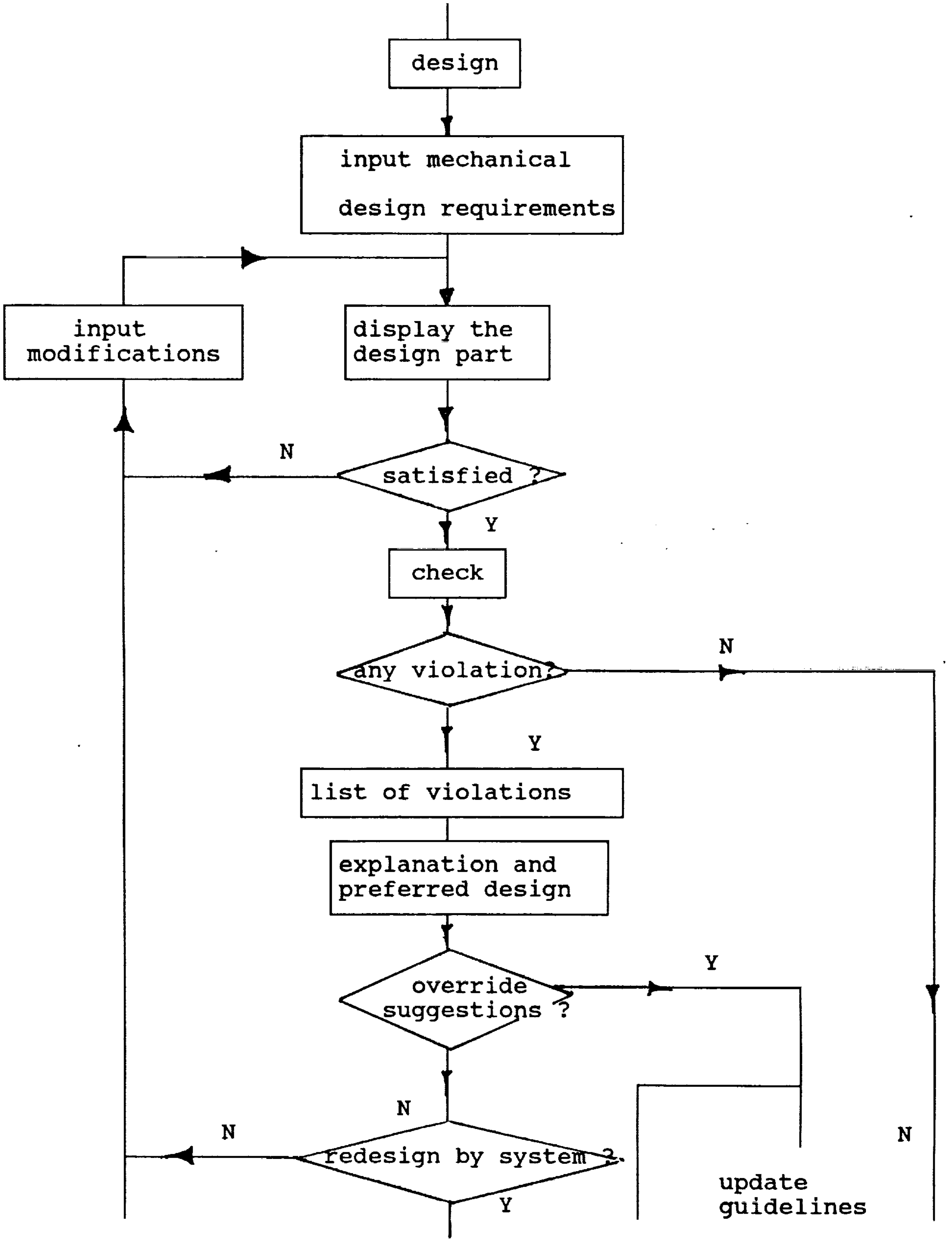


Fig. 2

B: The FUNCTIONS

The function of the prototype is shown in Fig. 3


design
input mechanical design requirements
input modifications
display the design part
N
satisfied ?
Y
check
any violation?
N
Y
list of violations
explanation and preferred design
override suggestions ?
Y
N
N
redesign by system ?
Y
N
update guidelines

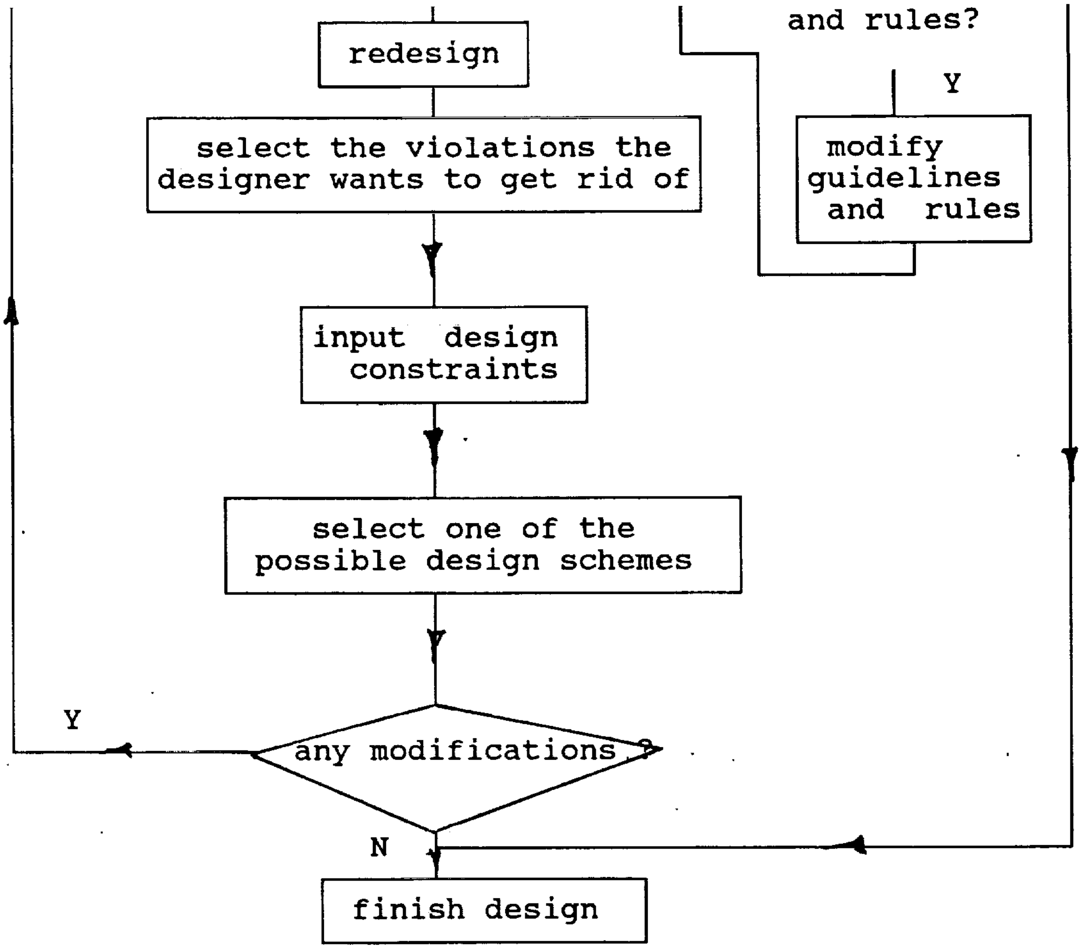

and rules?
Y
redesign
select the violations the designer wants to get rid of
modify guidelines and rules
input design constraints
select one of the possible design schemes
Y
any modifications ?
N
finish design

Fig. 3 system function diagram

a. Establish the database that can communicate with the AI module.

b. Start design by the user or by the system.

c. Output the design for the user to see if it is satisfied or not. If it is not satisfied, it can be modified either by the user or by the system.

d. Search the database to check the design. (There are two databases. One is the standard database, the other is the expanded database. Before the user expands the standard database, these two are actually one.) The user has the option to check his design against either the standard or the expanded database.

e. If there is any violation checked out, the user has 4 choices

   -- modify the design by himself

   -- modify the design by the system

   -- override these suggestions given by the system

   -- expand the existing database to make it more knowledgeable.

f. It is a loop (Fig. 4) during the design process. The loop will not terminate until the design is perfectly satisfied by the user.

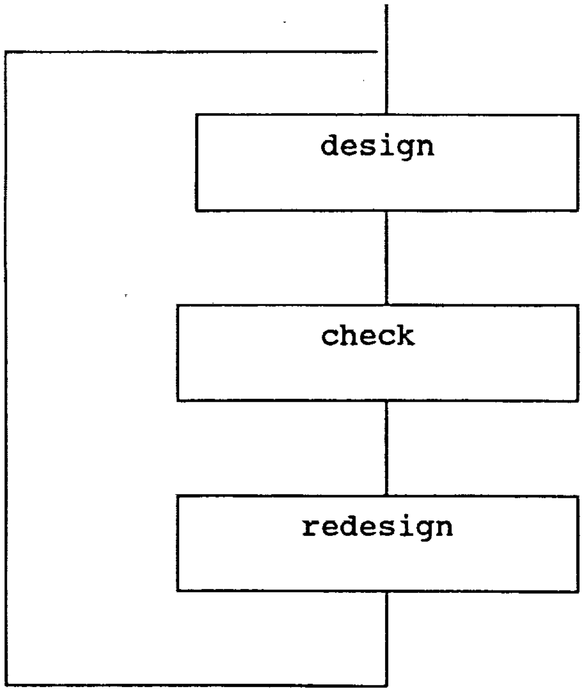


Fig. 4

C. Characteristics of this prototype:

(i) Frames are chosen to represent knowledge.

A major benefit of utilizing a frame structure is to make use of inheritance. When an entity is known to be a member of a class, its inherited properties can be assumed. This technique is especially meaningful for the mechanical parts design. For instance, the prototype we use here is a table. The same structure can also be applied in a bench or anything that has similar structure.

The frame of a table here has 4 slots. They are table_top, table_leg, leg_number and leg_variation.

The slot of the table_top contains information about top_shape, top_notch, top diameter if the table_top is round, the size of the two side if the top_shape is rectangular or square, the thickness of the top_shape, and the notch number.

The slot of table_leg contains information about leg_shape, leg_symmetry and leg length.

The slot of the leg_number contains information about the number of legs.

The slot of leg_variation contains information about whether all the legs have the same shape, same size or not.

(ii) knowledge base

The mechanical parts design is based on the knowledge and experience of expert practitioners of the task. In this prototype , we use 12 rules to simulate the guidelines of mechanical parts design. They are:

1: top shape (round or square)

2: diameter of a round table (20 -- 40)

3: short side of a rectangular table (10 -- 30)

4: long side of a rectangular table (20 -- 50)

5: thickness (1 -- 5)

6: table top should have notches for leg orientation

7: leg number equals to the notch number

8: number of legs (4)

9: all legs have the same size and the same shape to minimize the parts variation

10: leg shape (round or square)

11: leg length (25 -- 30)

12: legs are all symmetrical

Some of the rules are adopted from the guidelines in real world design. e.g. rule 6, 7, 9, and 12. So this prototype "clones" human experts to some degree.

(iii) Searching

This program helps user check his design against the database that stores human knowledge, expertise and experience. When the number of rules and guidelines exceeds thousand, the designer has no way but to rely on the computer's sophisticated searching techniques.

(iv) Decision making

With the feature of decision making, the design can be done or the design can be modified by the system based on the knowledge in the database.

In case the design is done by the system, the system would adopt the middle value of each parameter or the most commonly used value. In case the design requires modification by the system, the system would choose the lower bound value if the value chosen by the user exceeds the lower bound, or upper bound value if the value chosen by the user exceeds the upper bound (Appendix III).

CONCLUSION

We started building an expert system by the following steps:

1. Selected an application area -- "Mechanical parts design for outer space assembly".
2. Worked with mechanical experts to decide which guidelines and rules to be stored in the database.
3. Determined the techniques, knowledge and heuristics used by the experts for mechanical parts design.
4. Developed a prototype

4.1 Selection of software tool: two different approaches have been used. One is an "Exsys" software, while the other is "C" program.

4.2 Selection of hardware

4.3 Knowledge representation

4.4 Knowledge implementation

4.5 Testing and evaluation of the program

The above steps are essential to build expert system of any scale.

So far we have successfully developed a feasible prototype system. This design philosophy can be applied in our future work.

FUTURE WORK

We will develop a full prototype system for a reasonable and meaningful application. We will look for suitable

knowledge representation (could be frames). A suitable implementation approach would be to combine the software package with a generic "C" coding. This will reduce the development period, and will produce a reasonably flexible system with a high degree of efficiency.

REFERENCE

1. Advancing Automation and Robotics Technology for Space Station and for US Economy. NASA Technical Memorandum 87566. Submitted to US Congress, April, 1985.

2. Clarke M., Bronez, Mark A., "Telerobotics for the Space Station", Journal of Mechanical Engineering, February, 1986.

3. Robotics Servicer Workshop, Goddard Space Flight Center, Greenbelt, May, 1986.

4. Space Station Automation and Robotics Studies - Operator Systems Interface D483-10027-1, Boeing Co., 1984.

5. Space Station Automation Studies - Automation Requirements derived from Space Manufacturing Concept, Contract NAS 5-25182, 1984.

6. Automation Studies for Space Station Subsystgems and Mission Ground Support, Final Report submitted by Huge Aircraft Co., Reference No. F5713, Contract 82-14F, 1984.

7. NASA Space Station AI Based Technology Review Project 7268, SRI International, Menlo Park, CA, 1985.

8. Space Station Automation Study - Autonomous Systems and Assembly Contract NAS 8-35042, Martin Marietta, 1984.

9. Space Station Automation Study - Sattelite Servicing Contract NAS 8-35081. TRW Inc., 1984.

10. Nevins, J.L. and Whitney, D.E.:, Information and Control Issues of Adaptable Programable Assembly System for Manufacturing and Teleoperator Applications? Journal of Theory of Mechanisms and Machine Theory, Vol. 12, pp. 27-43, 1977.

11. Man Space Flight Study No. 981-10-30-04 Manipulator Systems for Space Application, Vol. I and II, Argan National Laboratory, April 1967.

12. NASA GFSC Plan for Space Station Robotics Program, Private Communication

13. Dwivedi, S.N., "Guidelines for Design for Manufacturability and Automation", Proc. of International Congress in Technology and Technology Exchange, Pittsburgh, Oct. 6-8, 1986.

14. Dwivedi, S.N. and Klein, B.R., "Design for Manufacturability Makes Dollar and Sense" Journal of CIM Review - A Journal of Manufacturing and Management, Vol. 2, No. 3, pp. 53-59, 1986.

15. Dwivedi, S.N., Gary Jones, and Travis, E. Use of CAD in Design and Development of Space Station and Robot Service" 2nd Int. Conf. on Robotics/Factories of Future.

16. Srivastava, S. and Sinha, V. "WEBXPERT - An Environment for Interactive Data Analysis" NASA TR 1986.

17. Srivastava, S. - The WSH SPACE (Budget System) - NASA TR 1983.

18. Srivastava, S. - Demonstration of Frame Base System - NASA TR 1982.

19. Srivastava, S. - Autonomous Scheduling Technology for Earth Orbital Missions - NASA TR 1980.

20. Srivastava, S. - Attitude determinations and Filters - NASA TR 1979.

Appendix 1

AN AI SUPPORTED SYSTEM FOR SPACE MECHANICAL PARTS DESIGN

ABSTRACT

This report outlines the work of the second phase of Automated Assembly in space. In the first phase, the guidelines and rules of mechanical design in space automated assembly has been discussed in detail. The second phase is the implementation of those principles.

The goal of this stage is to design and implement a system suitable for automated assembly in space. This system , in essence,is a combination of CAD and AI.

This report deals mainly with the function of the system, the architecture of the system and also the design issues of the system.

1. THE FUNCTION OF THE SYSTEM

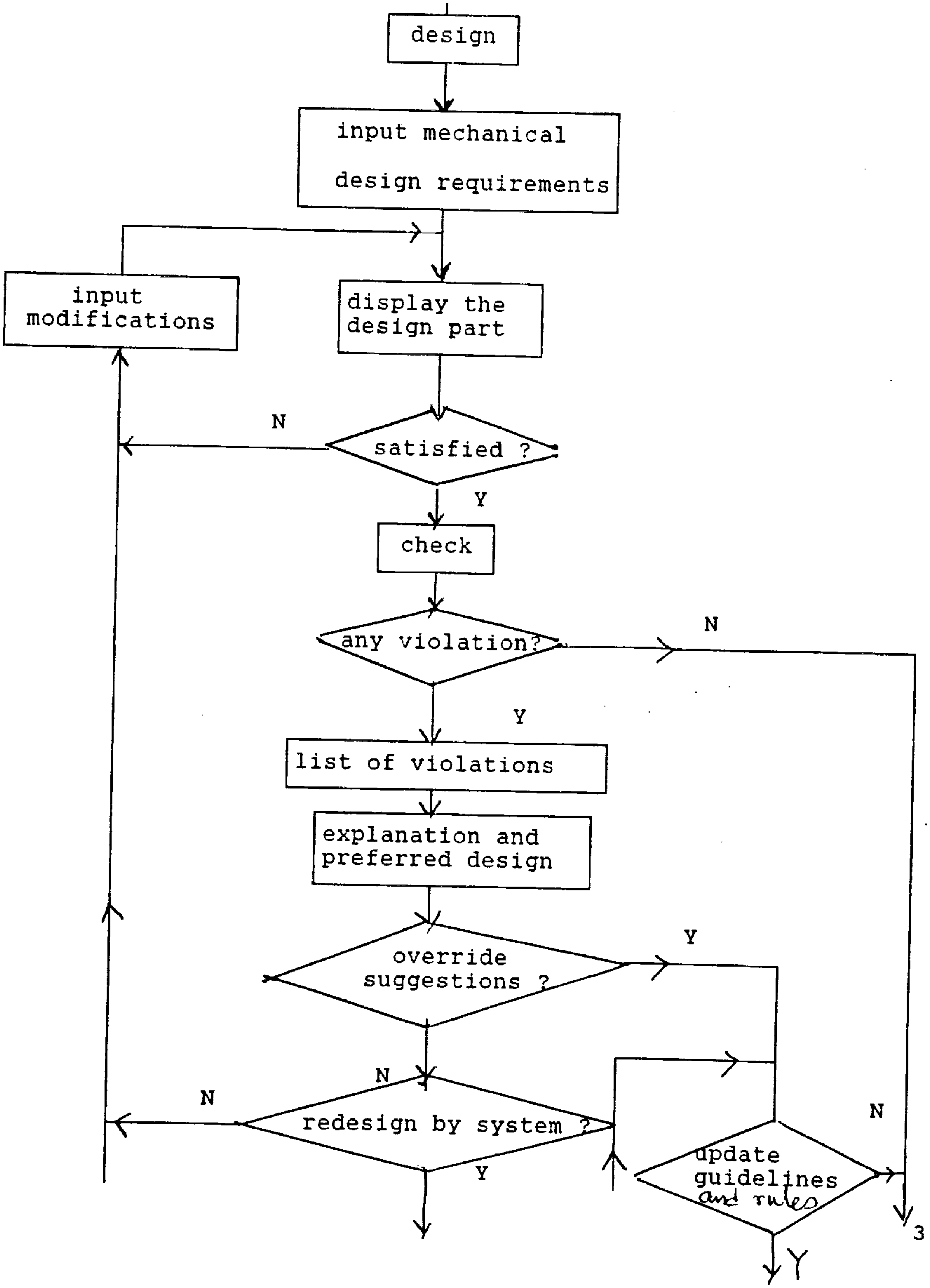
The user design the parts for space assembly using an interactive menu. The main menu includes the three major functions.

1. design
2. check
3. redesign

The first function "design" would invoke the 3D CAD design module for the user. After the user finishes the mechanical design, he would go back to the main menu.

Then the user can start the second function "check" to see if there are any violations of the rules for space assembly. If there is any, the system would give a list of violations for the user to check one by one. The function "check" also gives an explanation of each violation.

If the designer overrides these suggestions. He could modify the guidelines and rules. He could also choose to redesign either by himself or by the system. Of course, he could redesign the parts partially by himself and partially by the system.


design
input mechanical design requirements
input modifications
display the design part
satisfied ?
N
Y
check
any violation?
N
Y
list of violations
explanation and preferred design
override suggestions ?
Y
N
redesign by system ?
N
Y
update guidelines and rules
N
Y
3

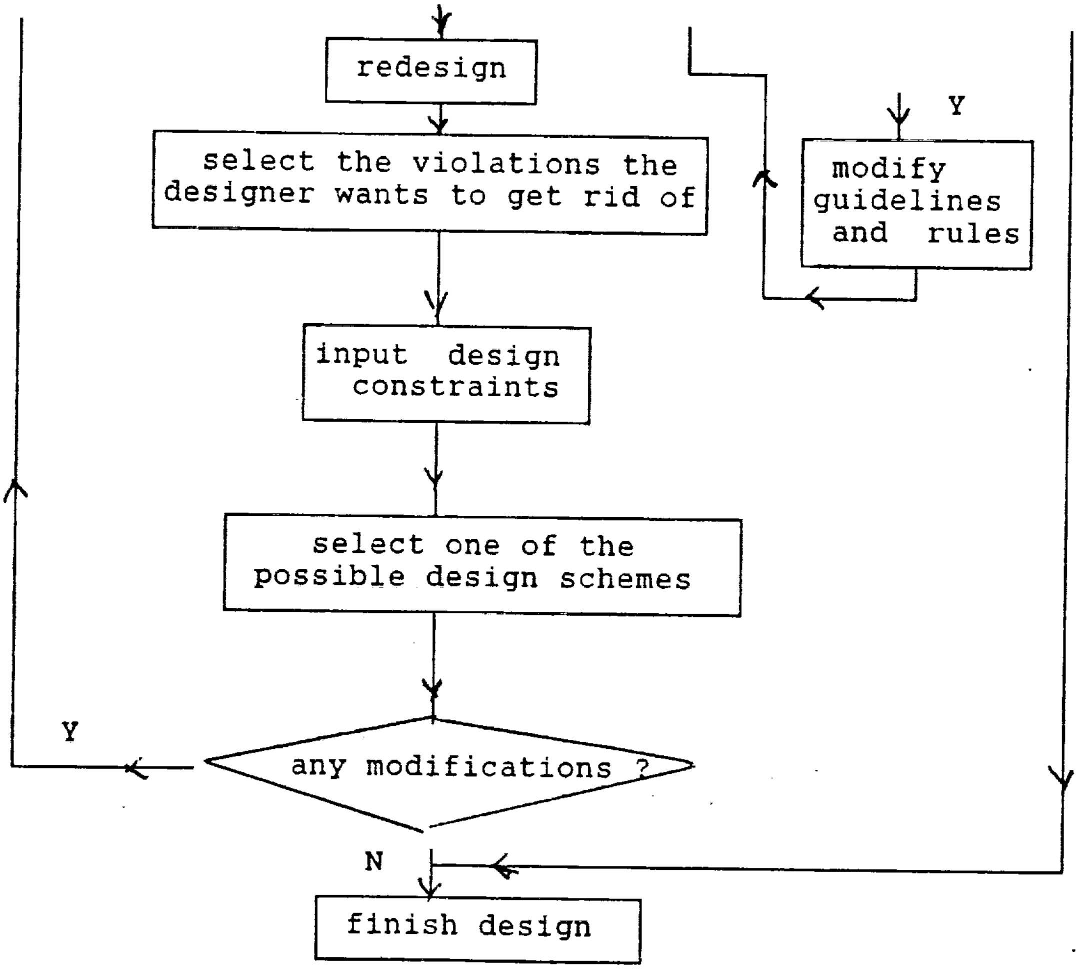

redesign
select the violations the designer wants to get rid of
Y
modify guidelines and rules
input design constraints
select one of the possible design schemes
Y
any modifications ?
N
finish design

Fig. 1 system function diagram

11. THE ARCHITECTURE OF THE SYSTEM

The system is composed by two major parts, i.e. the CAD and the AI. There are also an integrated data base and an interface to connect the two parts as illustrated in figure 2.

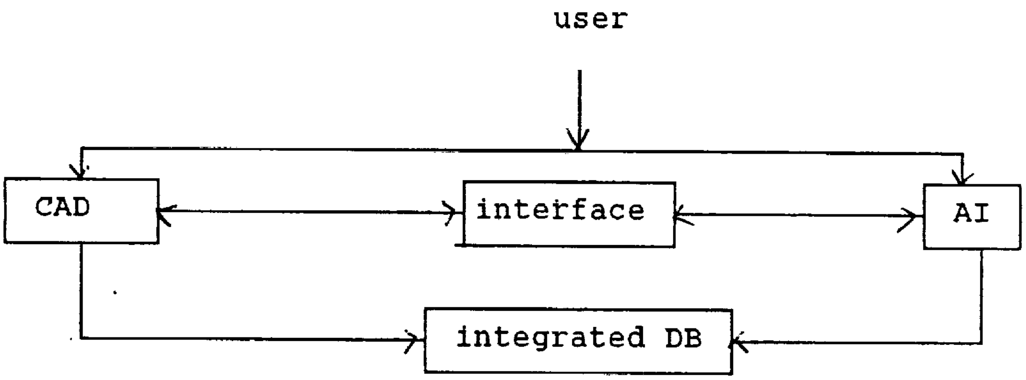


Figure 2. system structure

The AI in this system is actually an expert system which consists of four modules i.e. knowledge base, inference engine, explanation interface and knowledge acquisition. The structure is illustrated in figure 3.

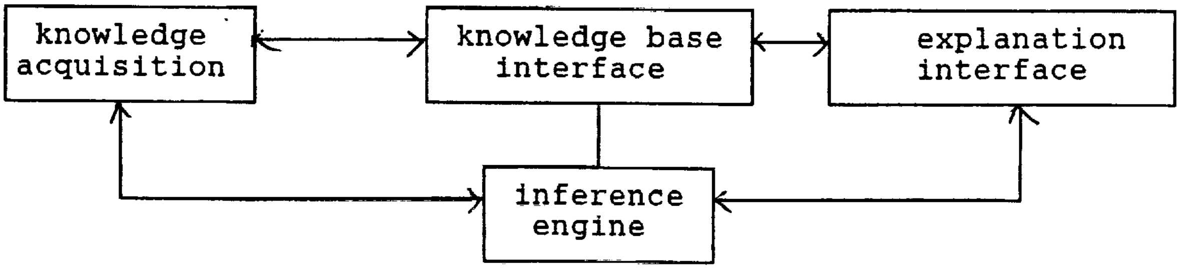


Figure 3. the structure of an expert system

Here are some brief explanation of the design goals of these four modules.

1. knowledge acquisition is a learning system that assimilates and accommodates knowledge by two ways. One is inputing by user while the other is learning from experience.
2. In knowledge base there is a knowledge base manager which manages facts and rules in the knowledge base.
3. The inference engine here is a Prolog system that serves as a general problem solving system.
4. The explanation interface gives user its advice and let the user know the inference procedure.

111. DESIGN ISSUES

1. Explanation interface

To design an explanation interface we have to write pieces of Prolog codes as an interpreter.

```
---------------------------------------------------------------

    1. solve (true).

    2. solve (A,B): - solve (A), solve (B).

    3. solve (A,B): - not solve (A).

    4. solve (not A): -not solve (A).

    5. solve (set_of (X,Goal, Xs)):- set_of(X,solve(goal),Xs).

    6. solve(A):- system(A),A.
```

```
7. solve(A):- clause(A,B),solve(B).
8. solve(A):-
          askable(A),not(known(A)),
          ask(A,Answer),
          respond(Answer,A).
9. ask(A,Answer):- display_query(A),
                   read(Answer).
10.respond(year,A):- asserta(A).
11.respond(no,A):- asserta(untrue(A)),fail.
12.known(A):- A.
13.known(A):- untrue(A).
14.display_query(A):- write(A),
                      write('?').
```

Clauses 1 to 7 can solve directly the user's goal. When the system cannot solve the problem, clauses 8 to 14 interactively talk with the user and continue to find the solution based on the information from the user. The system can also memorize these information. Each "yes" or "no" answer from the user would add a new fact to knowledge base that helps find the final goal.

A more complete system allows the user asks "Why?". When the system asks the user a question, the user can asks back the system why you have such a question. Then the system will list the rules that are the parent rules concerning this goal. If the user asks continuously, then the system will list the rules of rules that are the grandparent rules. If the user asks the system

"Why?" the third time, the system will list the great grandparent rules, so and forth. The corresponding codes are in appendix 1.

An even more complete system allows the user asks "How this goal was reached?". The system can explain that this goal was reached by using which rule and that rule was reached by another rule or fact, tracing back step by step until the user can understand the whole reasoning procedure. The corresponding codes are in appendix 2.

2. Knowledge representation

As all mechanical parts to be designed is the composition of geometric graphs, they are suitable to be represented by frame structure. We choose Prolog to implement the knowledge representation not only because Prolog has perfect reasoning but also because Prolog can check the consistency of the knowledge. One of the representation forms is

```
<Slot_Name>(<Frame_Name>,<Slot_Value>).
```

Let's take a simple example. A cylinder may be a component of a part. It can be described as

```
diameter (cylinder, a)
height (cylinder, b)
up_position (cylinder, c)
down_position (cylinder, d)
side_position (cylinder, e).
```

The operations of frame include:

(1). find: to find a slot_value for a given frame_name and slot_name.

(2). inheritance: some common property of son_frames can be described by their father_frame. It is not necessary for the son_frames to list all the properties of father_frame, because son_frames automatically inherit all the properties of the father_frame.

(3). update: to update the slot_value.

(4). demo processing: demo processing can automatically calculate, add or delete a slot_value. For instance, if a symmetric part has two holes, the size of one hole is unknown. When we try to find the size of it, the demon procedure will be invoked automatically to give the estimated size by the property of symmetry. The procedures for default values are mostly written on the basis of experience.

(5). default value: the slot_value of the son_frames can be default.

Here we introduce some very useful predicates for knowledge representation. They can tremendously increase Prolog representation ability.

a. select: to select an appropriate subgoal from a group of goals.

b. find: to select a rule from a group of rules.

c. match: to find the matching methods and matching requirements.

d. merge: to combine the subsequent goal with the rest of goals as a new goal.

e. react: to find a rule based on the subgoal together with a certain strategy that the rule_head must be matched with the subgoal.

f. demo: to prove the goal.

g. name_of: to give the strategy of finding rules.

h. is_a: to store the slot_value.

i. update: to update the slot_value.

j. drop_from: to get rid of old frame.

k. add_to: to establish new frame.

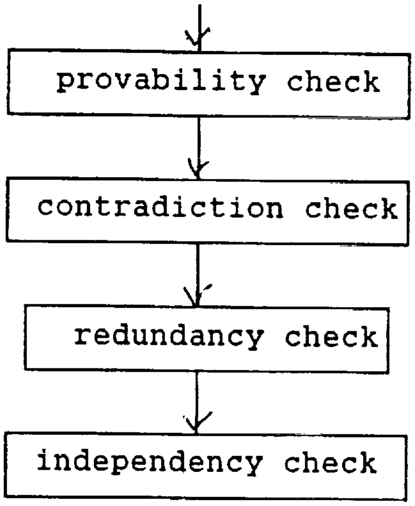
Appendix 3 gives the code of demo. It is very useful even in knowledge acquisition module.

3. Knowledge acquisition

The flexibility of the knowledge base allows experts to add or modify the guidelines and rules so as to upgrade the system ability. It is implemented by the module of knowledge acquisition. Knowledge acquisition has two major functions, one is knowledge assimilation while the other is knowledge accommodation.

(1). Knowledge assimilation: If and only if the knowledge is necessary and compatible with the purpose of establishing the knowledge base, then the knowledge can be stored into the knowledge base. In such a case, we always assume that the knowledge in knowledge base is always correct and can not be modified.

The steps of knowledge assimilation is as follows:



(2). Knowledge accommodation: In this case, we assume that the input knowledge is correct. We use the input knowledge to check and update the knowledge which has already been in the knowledge base.

We define the following predicates to implement the knowledge acquisition.

a. input: to input knowledge

b. currdb: the current database

c. ic: the integrity constraints

d. check_db: to check the input to see if it is assimilable

e. assimilate: to assimilate the input knowledge

f. deduce: to form a deduction mechanism

The corresponding codes are in appendix 4.

4. Search implementation

We use a goal stack and a planning stack to implement goal driven reasoning.

(1). debugging tools: There are three cases that might occur during search processes.

a. infinite loop or recurrsion

b. reach false goal

c. miss solution

So we define the following predicates as tools in our search

(a). solve: to terminate the program when the stack is overflow

(b). false_solution: to delete or modify a clause when the falsity is detected from the provable tree

(c). miss_solution: to trace the failure path so as to find a goal that is not covered by the program

The corresponding codes are in appendix 5,6,7.

(2). Model reasoning: Model reasoning includes induction mechanism. It searches the possible collection of clauses. If a clause cannot cover a goal, it is cut off together with its branch. It continues to search until the final goal is found.

The corresponding codes are in appendix 8.

3. Heuristic search: As all mechanical parts to be designed

are geometric graph, we can use graph information to implement heuristic graph search.

Let's take just a simple example.

Suppose we are given the theorems about two equal triangle.

```
(1) tri (A,B,C) = tri (D,E,F):-
          angle (A,B,C) = angle (D,F,E),
          angle (C,A,B) = angle (F,D,E),
          seg (B,C) = seg (E,F).
(2) seg (A,B) = seg (D,E):-
          tri (A,B,C) = tri (D,E,F).
(3) seg (A,B) = seg (D,E):-
          seg (A,B) = seg (E,D).
```

The two triangles are as follows:

Now, if we want to prove that the distances from any point in the line that equally divides an angle to the two sides of that angle are equal.

See the following graph:

So we are given the message:

```
(1) angle (d,a,b) = angle (d,c,b)
(2) angle (a,b,d) = angle (c,b,d)
```

We want to prove:

:-seg (a,d) = seg (c,d)

The search tree would be structured as follows:

```
                         :-seg (a,d) = seg (c,d)
        theorem (2)                                  theorem (3)


:-tri(a,d,X)=tri(c,d,Y)                         :-seg(a,d)=seg(d,c)


theorem(1)                                               theorem(2)


:-angle(a,X,d)=angle(c,Y,d),                  :-tri(a,d,Z)=tri(d,c,W)
  angle(d,a,X)=angle(d,c,Y),                         ( * * )
  seg(d,X)=seg(d,Y).
               ( * )
```

We cannot go further without using graphic information. However, taking graphic information as heuristic information, we find the substitution of X and Y with b, but there is no substitution for Z and W that tri(a,d,Z)=tri(d,c,W). After cutting off (* *) branch, we minimize our search greatly.

The corresponding codes for heuristic search are in appendix 9.

lV. Conclusion

We've discussed an AI supported CAD system, its function, its architecture as well as some major design issues which are the cornerstones of the system. We also give relevant codes for the implementation of the system.

V. References

[1] Begg, Vivienne. Developing expert CAD systems /. London, Kogan Page, c1984.

[2] Holden, Tony. Knowledge based CAD and microelectronics /. Amsterdam New York, North-holland New York, N.Y., U.S.A, Sole distributors for the U.S.A. and Canada, Elsevier science Pub, 1987.

[3] Luger, George F. Artificial intelligence and the design of expert systems /. Redwood City, Calif, Benjamin/Cummings Pub. Co, c1989.

[4] Merritt, Dennis. Building expert systems in Prolog /. New York, Spring-verlag, c1989.

Appendixes

Notes of appendixes:

1. The following codes demonstrate the design logic of the author concerning the design issues.
2. None of the parameters are the actual parameters. But many of their names are meaningful so as to help the reader to understand these codes.
3. These codes are used in actual programming design with only a little modification.

APPENDIX 1

```
solve(Goal):-solve(Goal,[]).

solve(true,Rules).

solve((A,B),Rules):-solve(A,Rules),solve(B,Rules).

solve((A;B),Rules):-solve(A,Rules);solve(B,Rules).

solve(not(A),Rules):-not(solve(A,Rules)).


solve(set_of(X,Goal,Xs),Rules):-

          set_of(X,solve(Goal,Rules),Xs).


solve(A,Rules):-system(A),A.


solve(A,Rules):-clause(A,B),solve(B,[rule(A,B)|Rules]).


solve(A,Rules):-

     askable(A),not(known(A)),

     ask(A,Answer),

     respond(Answer,A,Rules).


ask(A,Answer):-

     display_query(A),

     read(Answer).


respond(yes,A,Rules):-asserta(A).
respond(no,A,Rules):-asserta(untrue(A)),fail.

respond(why,A,[Rule | Rules]):-
     display_rule(Rule),
     ask(A,Answer),
     respond(Answer,A,Rules).

respond(why,A,[]):-
     write('No more explanation possible'),nl,
     ask(A,Answer),
     respond(Answer,A,Rules).

known(A):-A.

known(A):-untrue(A).

display_query(A):-write(A),write('?').

display_rule(rule(A,B)):-
     nl,write('IF'),
     write_conjunction(B),
     write('THEN'),write(A),nl.

write_conjunction((A,B)):-
     !,write(A),write('AND'),
     write-conjunction(B).
```

```
write_conjunction(A):-write(A),nl.
```

APPENDIX 2

```
how(Goal):-
     solve1(Goal,Proof),interpret(Proof).

solve1(true,true).

solve1((A,B),(ProofA,ProofB)):-
     solve1(A,ProofA,solve1(B,ProofB).

solve1(A,(A<--Proof)):-
     clause(A,B),solve(B,Proof).

interpret((Proof1,Proof2)):-
     interpret(Proof1),interpret(Proof2).


interpret(Proof):-
     fact(Proof,Fact),
     nl,write(Fact),
     write('is a fact in the database'),nl.

interpret(Proof):-
     rule(Proof,Head,Body,Proof1),nl,
        write(Head),write('is proved using the rule'),nl,
        display_rule(rule(Head,Body)),
        interpret(Proof1).

fact((Fact<--true),Fact).

rule((Goal<--Proof),Goal,Body,Proof):-
        Proof\=true,
        extract_body(Proof,Body).

extract_body((Proof1,Proof2),(Body1,Body2)):-
        !,
        extract_body(Proof1,Body1),
        extract_body(Proof2,Body2).

extract_body ((Goal<--Proof),Goal).
```

APPENDIX 3

```
demo(Theory,Goal,[]):-empty(Goal).

demo(Theory,Goal,[Reason|Rest_Proof])
```

```
     :-
     select(Goal, Subgoal, Rest_Goals),
     react(Theory, Subgoal, Reason, Continuation_Goals),
     merge(Continuation_Goals, Rest_Goals, New_Goal),
     demo(Theory, New-Goal, Rest_Proof).

react(Theory, demo(New-Theory, Subsid-Goal, Subsid_Proof),
       sbs(Subsid_Proof), true)
     :-
     demo(New_Theory, Subsid-Goal, Subsid_Proof).

react(Theory, current(Theory), current(Theory),true).

react(Theory, Subgoal, s(Subgoal, Rule), Rule_Body)
     :-
     find(Subgoal, Theory, Rule),
     parts(Rule, Rule-Head, Rule_Body),
     match(Subgoal, Rule_Head).
```

APPENDIX 4

```
assim(KBNL,Input):-
          not(assim_ka1(KBNL,Input)),
          not(assim-ka2(KBNL,Input)),
          not(assim_ka3(KBNL,Input)),
          assim_ka4(KBNL,Input).
```

```
/*  K1  Deducility  check  */

assim_ka1 (KBNL,Input) :-

     deduce (KBNL, Input, _) , nl,

     write ('Input:'), write (Input),nl,

     write ('Input_Knowledge is deducible from DB! '),nl.


/* K2  Contradiction Check (against integrity constraint) */

assim_ka2 (KBNL,Input):- X=..[KBNL,Input], assertz (X),

     G=..[check_db, Input, IC, Mes, KBNL], Y=..[KBNL, G],

          Y, ic_trans (IC, Icr) ,

          demo (KBNL, Icr, [[], Cond, Cut, 50], [true,[]]),

          write ('Input context:'), write (Input),

         nl,   write   ('Conflicts   with   the   integrity

          constraint!'), nl,

          write (Mes), nl, retract (X).


assim_Ka2 (KBNL, Input):- X=..[KBNL, Input],

          retract (X), fail.


: - op (205, xfx, ['- - - >']).


ic_trans((Icp- - - >  Icq), (Icp, not (Icq))).


ic-trans((Ica; Icb), (Icam, Icbm)):-

          ic-trans (Ica, Icam) , ic-trans (Icb, Icbm) .
```

```
ic_trans(((Icp - - - >Icq), Icb), (Icq, not (Icq) ;Icbm)):-
          ic_trans (Icb, Icbm).


/*  K3  Redundancy check in (Db + Input) */
assim_ka3 (KBNL, Input):-
          Y=..[KBNL,X], Y, noredun (X, Input, KBNL), fail.


noredun((P:- Q), Input, KBNL) :-!, fail.


noredun (X, Input, KBNL): -
          Y=..[KBNL, X], Z=..[KBNL, Input], retract(Y),
          asserta(Z), demo (KBNL, X, [[True],-,-,50][-,-]),
          retract (Z), ! .


noredun (X, Input, KBNL):-
          Y=..[KBNL, X], Z=..[KBNL, Input], retract (Z),
          asserta (Y), !.


/*  K4  Independency check  */
assim-ka4 (KBNL, Input):- Z=..[KBNL, Input], assertz (Z),nl,
          write ('new knowledge:'), write (Input),
          write (' is acquired! '), nl.
```

APPENDIX 5

```
solve (true, D, no_overflow).
solve (A, 0, overflow ([])).

solve ((A,B), D, Overflow):-
     D>0,
     solve (A, D, Overflow A),
     solve_conjunction (OverflowA, B, D, Overflow).

solve (A, D, no-overflow) :-
     D>0,
     system (A), !, A.

solve (A, D, Overflow) :-
     D>0,
     clause (A,B),
     D1 is D-1,
     solve (B, D1, OverflowB) ,
     return_overflow (OverflowB, A, Overflow).

solve_conjunction (overflow (S) , B, D, overflow (S)).
solve_conjunction (no_overflow, B, D, Overflow) :-
     solve (B, D, Overflow).

return_overflow (no_overflow, A, no-overflow).
return_overflow (overflow (S) , A, overflow ([A | S])).

-----------------------------------------------------------------
```

APPENDIX 6

```
false_solution (A, Clause):-
     solve (A, Proof),
     false_goal (Proof, Clause).
false_goal ((A :- B), Clause):-
     false_conjunction)(B, Clause), !.

false_goal ((A :- B), (A :- B1)):-
     extract_body (B, B1).


false-conjunction ((A :- B), Bs), Clause):-
     query_goal (A, false) ,! ,
     false_goal ((A :- B), Clause).

false-conjunction ((A :- B), Clause):-
     query_goal (A, false) , ! ,
     false_goal ((A :- B) , Clause) .

false_conjunction ((A, As), Clause):-
     false_conjunction (As, Clause).
```

APPENDIX 7

```
missing_solution ((A, B), Goal):- !,
     (not (A), missing_solution (A, Goal);
     A, missing_solution (B, Goal)).

missing_solution (A, Goal):-
     clause (A,B), query_clause ((A :- B)), ! ,
     missing-solution (B, Goal).

missing_solution (A, A):- not (system (A)).

query_clause (Clause):-
     write ('Enter a true ground instance of'), nl,
     write (Clause) , nl,
     write ('if there is such, or "false" otherwise') ,nl,
     read ( Answer), ! , check_answer (Answer, Clause) .

check_answer (false, Clause):- !, fail.

check_answer (Clause, Clause) :- ! .

check_answer (Answer, Clause) :-
     write ('illegal answer'), ! , query_clause (Clause).
```

APPENDIX 8

```
model_inference (KBN, IC_name, Concept):-
     nl,
     ask_for('Next fact(sentence, true/false)or end',Fact),
     (Fact=end;
     (Fact=check -> model_inference1 (KBN, Ic_name,-);
     (Fact= (P,V), verify (P=Concept);
     nl,
     write ('* Error concept from.'), fail),
     !,
     (Fact= (P,V),
     (V=true; V=false) ->
          assert_fact (P,V),
     model_inference (KBN, Ic_name, P),
     write ('! illegal input') , nl), ! , nl,
     model_inference (KBN, IC_name, Concept))).


model_inference (KBN, IC_name, Concept):-
     nl,
     ask_for('Next fact(sentence, true/false)or end',fact),
     (Fact=end;
     (Fact=check -> model_inference1(KBN,Ic_name,-);
     (Fact=(P,V), (V=true; V=false) ->
     assert_fact (P,V),
     model_inference1 (KBN, Ic_name, P);
```

```
          write (' ! illegal input'), nl), ! , nl,
          model_inference (KBN, IC_name, Concept))).

     model_inference1 (KBN, IC_name, P) :-
          write ('checking  fact (s) ...'),
          (fact (P, false),
          model-demo (KBN, P) ->
          nl, false_solution (KBN,IC_name, P),
          model_inference1 (KBN, IC_name, _);
                              /* Too Strong */
          (fact (P, true),
          not (model_demo (KBN, P))  ->
          nl, missing_solution (KBN, IC_name, P),
          model_inference (KBN, IC_name, _),
                              /* Too Weak */
          write ('no error found.') , nl)).

     verify (P):- not (not(P)).

     ask_for (Mess, Fact):-
          write (mess), nl, read (Fact) .
```

APPENDIX 9

```
/*  Heuristic Search THEOREM PROVER  */
```

```
go:- asserta(step(0), heuristic([pair(1,goal)]).


/*  Top Level Stuff */

heuristic (Agenda):-

     member(pair(0, Empty), Agenda).


heuristic([pair(Score, Current) | Rest]):-

     step(X),

     X1 is X+1,

     retract(step(_)),

     X<10,

     asserta(step(X1)),

     nl, write('step number is:'), write(X1),nl,

     settof1(Clause,successor(Current,Clause),New Clause),

     add_to_agenda(New Clause, Rest, New Agenda),

     heuristic(New Agenda).


successor(Current, Clause):-

     successor1(Current, Clause1),

     vet(Clause1, Clause, model1),

     message(Clause1).


successor1(Current, Clause):-

     factor(Current, Clause).


successor1(Current, Clause):-
```

```
          is_clause(Parent,_,_),
          (resolve(Current, Parent, Clause);
          resolve(Parent, Current, Clause)).

/* Rules of Inference */
resolve(Parent1, Parent2, Resolvant):-
          is_clause(Parent1, Consequent1, Antecedent1),
          is_clause(Parent2, Consequent2, Antecedent2),
          select(Proposition, Consequent1, RestConse1),
          select(Proposition, Antecedent2, RestAnte2),
          append(RestConse1, Consequent2, Consequent),
          append(Antecedent1, RestAnte2, Antecedent),
          gensym(resolvant, Resolvant),
          assertz(is_clause(Factor, OneGone; Antecedent)).

factor(Clause, Factor):-
          is_clause(Clause, Consequent, Antecedent),
          select(Proposition, Consequent, OneGone),
          select(Proposition, OneGone, TwoGone),
          gensym(factor, Factor),
          assertz(is_clause(Factor, OneGone, Antecedent)).

factor(Clause, Factor):-
          is_clause(Clause, Consequent, Antecedent),
          select(Proposition, Antecedent, OneGone),
          select(Proposition, OneGone, TwoGone),
     gensym(factor, Factor),
     assertz(is_clause(Factor, Consequent, OneGone)).




/* Print Message */
message(Clause):-
     is_clause(Clause, Conse, Ante),
     write('New Clause'),
     write(Clause),
     write('is:'),
     write(Ante),
     write('->'),
     write(Conse), nl.

/* Evaluation Function */
add_to_agenda([], Agenda, Agenda).
add_to_agenda([Name | Rest], Agenda, NewAgenda):-
     (not(in(Name)), !; true),
     evaluate(Name, Score), !,
     insert_into_agenda(Agenda, Score, Name, MidAgenda),
     add_to_agenda(Rest, MidAgenda, New Agenda).

add_to_agenda([Name | Rest], Agenda, NewAgenda):-
     add_to_agenda(Rest, Agenda, NewAgenda).
```

```
evaluate(Name, Score):-
     is_clause(Name, Consequence, Antecedent),
     length(Consequence, C),
     length(Antecedent, A),
     Score is C + A.


insert_into_agenda([], Score, Name, [pair(Score, Name)]).
insert_into_agenda([pair(Score1, Name1)| Rest], Score,
  Name, [pair(Score, Name), pair(Score1, Name1)|Rest]):-
     Score=<Score1,
     !.


insert_into_agenda([X|Rest], Score, Name, [X | NewRest]):-
     insert_into_agenda(Rest, Score, Name, NewRest).


/* Loop Check */
in(Clause):-
     is_clause(Clause, Conse, Ante),
     is_clause(Another, Conse, Ante),
     Clause\= = Another,
     retract(is_clause(Another, Conse, Ante)).


/* Semantic Checking */
vet(Clause1, Clause, Interp):-
     is_clause(Clause, Conse, Ante),
     constants(Consts),
     checklist(instantiate(Consts), Conse),
     checklist(instantiate(Consts), Ante),
     false_clause(Conse, Ante, Interp),
     gensym(instance, Clause),
     assertz(is_clause(Clause, Conse, Ante)).

instantiate(Consts, Constant):-
     atomic(Constant).

instantiate(Consts, Variable):-
     var(Variable), member(Variable, Consts).
instantiate(Consts, Complex):-
     not(atomic(Complex)),
     novar(Complex),
     Complex=..[Sym | Paras]
     checklist(instantiate(Consts), Paras).

false_clause(Consequent, Antecedent, Interp):-
     checklist(meaning(interp, false), Consequent),
     checklist(meaning(Interp, true), Antecedent).

meaning(Interp, Value, Constant):-
     atomic (Constant),!,
     e(Interp, Value, Constant).

meaning(Interp, Value, Complex):-
```

```
    Complex=..[Sym | Paras],!,
    maplist(meaning(Interp), Vals, Paras),
    Complex1=..[Sym | Vals],
    interpret(Interp, Value, Complex1).

/* General Utilities */
append([], List, List):-!.
append([Car | Cdr], List, [Car | Ans]):-
    append(Cdr, List, Ans).

member(E,L):-
    append(L1, [E | L2], L).

select(E,L,R):-
    append(L1, [E | L2], L),
    append(L1, L2, R).

/* Logical */
setof1(X, P, Set):- setof(X, P, Set),!.
setof1(X, P, []).

checklist(P, []):-!.
checklist(P, [Y | YList]):-!,
    P=..[Sym | XList],
    append(XList, [Y], Paras),
    Q=..[Sym | Paras],
     Q,
     checklist(P, YList).

maplist1(P,[].[]):-!.
maplist1(P, [Y | YList], [Z | ZList]):-!,
     P=..[Sym | XList],
     append(XList, [Y,Z], Paras),
     Q=..[Sym | Paras],
     Q,
     maplist(P, YList, ZList).

     /* Generate New Name */

gensym(Prefix, Var):-
     var(Var), atomic(Prefix),
     get(Prefix, N),
     N1 is N+1,
     asserta(latest(Prefix, N1)),
     concat(Prefix, N1, Var).

get(Prefix, N):- retract(latest(Prefix, N)),!.
get(Prefix,0).

concat(N1, N2, N):-
     name(N1, Ls1),
     name(N2, Ls2),
      append(Ls1, Ls2, Ls),
      name(N, Ls).

/* Example Specific Stuff */

/* Axioms of Negated and Conjecture */

is_clause(reflexive, [equal(X,X)], []).
is_clause(funny, [equal(X, Y)], [equal(X, z),equal(z, Y)]).
is_clause(twisted, [equal(U,W)], [equal(U,V), equal(W,V)]).
is_clause(hypothesis, [equal(x,y)], []).
is_clause(goal, [], [equal(y,x)]).

/* Constants */

Constants([x,y,z]).

/* Interpretation Model1 */

interpret(model1, 2, x).
interpret(model1, 2, y).
interpret(model1, 3, z).
interpret(model1, true, equal(X, Y)):-
     X= =Y
interpret(model1, false, equal(X, Y)):-
     X\= =Y.
```

Appendix 2

EXSYS: building an expert system.

EXSYS has a wide range of capabilities and can be used for building expert system. EXSYS allows you to use both text qualifier/value type conditions, such as we used in lesson 1, and also mathematical expressions with variables. Math/variable expressions in EXSYS cover several areas. There can be both numeric and string variables. A variable can have a value associated with it or it can be a text note. Trigonometric, exponential, log and square root functions are built into the program.

There are many ways in which EXSYS allows to edit rules. The parameter in the expert system can be changed. One exception is the probability mode, it cannot be changed after it has been selected once. The editing is not very complicated. There are three part screen. On the left hand side are the rules. On the right is the qualifier. At the bottom are the choices for the operation on the. Rules can be added, deleted or modified. The same with the qualifiers and variables. After adding rules it can be tested by choosing the "run" option. The "store/exit" option saves data on the desired drive. The directions are very straight forward and easy to follow.

EXSYS is capable of doing a wide range of calculational tasks, but there are some things beyond its capabilities. EXSYS has the ability to call external programs to do such work. The external program may pass data to EXSYS from a data base or spreadsheet, run a custom program fro graphics or calculations or obtain data or obtain data from automatic text equipment.

EXSYS has the ability to call almost any MS-DOS program as an external program. When an external program is called, EXSYS remains resident in the memory and the external program is loaded in the higher memory. When the external program is finished and exits as if it were going to DOS, it returns instead to EXSYS.

External programs return data to EXSYS by writing it to a disk file in ASCII. THis enables EXSYS to communicate with the largest number of programs to communicate with the largest number of programs and the external program is loaded in the higher memory. When the external program is finished and exits as it it were going to DOS, it returns instead to EXSYS.

External programs return data to EXSYS by writing it to a disk file in ASCII. This enables EXSYS to communicate with the largest number of programs. The format of the returned data depends on the type of the external call. There can be a call to an external program at the start of a run that can return data for multiple qualifiers or variables. Such data is returned is returned as a Q or V, followed by the number of qualifier or variable. If the external program is tied to a single qualifier or variable.

The data is returned by writing it to a file called RETURN.DAT. You also have the option of changing the name of this file if your external program has particular file names it prefers. Also you may wish to pass the data in a file in RAM disk, which greatly increases the speed.

It is the responsibility of the external program to write the data in the correct format in the correct file. EXSYS will automatically look for the file and read the data.

You can also pass the data out to an external program. The data can be passed out by writing it to a disk file in ASCII or it can be passed as a command line argument (only some programs can accept data passed as command line arguments). However, the demo package is not capable of passing any parameters.

EXSYS or similar expert system shells provides an easy way to input the rules, thus creating the knoledge base. EXSYS is rule based. The actual EXSYS has far greater capabilitiesand is suitable for creating expert systems. With that it is possible to have a gigantic knowledge base, thus far better expert system. Overall these shells save lot of time and money, as one does not have to create the expert system from scratch. It would be great experiance to work with the EXSYS "real" shell.

Attached are two examples of mini expert systems-TABLE EXPERT.

ORIGINAL PAGE IS
OF POOR QUALITY

Subject:
TABLE EXPERT

Author:
YOUSUF AHMAD

Starting text:

WELCOME

TO

TABLE

EXPERT

The TABLE EXPERT makes recommendations whether or not your supplied data can be used to design a square or circular table. On some occasions the system may even request you to update the rules or qualifiers or even the values.

Ending text:

RECOMMENDATIONS

Based on the answers provided the TABLE EXPERT makes the following recommendations:

Uses all applicable rules in data derivations.

VARIABLES:

1 TOP SHAPE
type of top shape of the table
String variable

2 LEG SHAPE
shape of table legs
String variable

3 LENGTH EACHSIDE
length of each side
Numeric variable

4 NUMBER OF NOTCHES
number of notches on table to fit the legs
String variable

5 LENGTH1
length of each side
Numeric variable

6 LENGTH EACHLEG
length of each leg
Numeric variable

7 NOTCHES
number of notches
Numeric variable

8 LEG LENGTH

Numeric variable

9 NUMBER OF LEGS
number of legs of the table
Numeric variable

10 COUNTER
keeps count of successes
Numeric variable

11 DIAMETER
the diameter of the table
Numeric variable

```
RULES:


----------------------------------------



RULE NUMBER: 1

IF:
       designer is user
  and  table is square


THEN:
       dimension key is one side
  and  [TS] IS GIVEN THE VALUE [TS]


----------------------------------------



RULE NUMBER: 2

IF:
       dimension key is one side


THEN:
       [L1] IS GIVEN THE VALUE [L1]
  and  design objective is square


----------------------------------------



RULE NUMBER: 3

IF:
       [NL] <= 6
  and  dimension key is one side


THEN:
       design objective is square


----------------------------------------



RULE NUMBER: 4

IF:
   and  [TS] = "SQUARE"
   and  dimension key is one side
   and  design objective is square
   and  [N] = [NL]


THEN:
        You can design a square table - Probability=1


----------------------------------------



RULE NUMBER: 5

IF:
        designer is user
  and   table is circular


THEN:
        dimension key is diameter
  and   [TS] IS GIVEN THE VALUE [TS]


----------------------------------------



RULE NUMBER: 6

IF:
        dimension key is diameter


THEN:
        design objective is circular
  and   [D] IS GIVEN THE VALUE [D]


----------------------------------------



RULE NUMBER: 7

IF:
        [NL] <= 6
  and   dimension key is diameter


THEN:
        design objective is circular


----------------------------------------



RULE NUMBER: 8
```

```
IF:
       designer is user
  and  [TS] = "CIRCULAR"
  and  [N] = [NL]
  and  dimension key is diameter
  and  design objective is circular

THEN:
       You can design a circular table - Probability=1

----------------------------------------



RULE NUMBER: 9

IF:
       [TS] <> "SQUARE"
  and  table is square

THEN:
       Your choice of topshape is invalid - please try again - Probability=1

----------------------------------------



RULE NUMBER: 10

IF:
       [TS] <> "CIRCULAR"
  and  table is circular

THEN:
       Your choice of topshape is invalid - please try again - Probability=1

----------------------------------------



RULE NUMBER: 11

IF:
       [N] <> [NL]

THEN:
       Number of notches does not match the number of legs - Probability=1

----------------------------------------



RULE NUMBER: 12
```

```
IF:
        [TD] = "SYSTEM"


THEN:
        designer is system

-----------------------------------------



RULE NUMBER: 13

IF:
        designer is system
  and   [U] = 0


THEN:
        get info from last user


ELSE:
        get info from second last user

-----------------------------------------



RULE NUMBER: 14

IF:
        [TD] = "USER"


THEN:
        designer is user
```

```
QUALIFIERS:



1    shape of leg is

  circular
  not circular


       Used in rule(s):




2    table is

  square
  circular


       Used in rule(s):     1       5       9       10




3    length of each side does

  exist
  not exist


       Used in rule(s):




4    design objective is

  square
  circular


       Used in rule(s):  (   2)  (   3)      4   (   6)  (   7)      8




5    dimension key is

  one side
  diameter


       Used in rule(s):  (   1)      2       3       4   (   5)      6
```

```
6    designer is

  user
  system


       Used in rule(s):      1       4       5       8   (  12)     13
                         (  14)



7    RUN(DATAGEN) get info from

  last user
  second last user


       Used in rule(s):  (  13)  [  13]
```

```
CHOICES:

1     You can design a square table

         Used in rule(s):  (   4)


2     You can design a circular table

         Used in rule(s):  (   8)


3     Your choice of topshape is invalid - please try again

         Used in rule(s):  (   9)  (  10)


4     Number of notches does not match the number of legs

         Used in rule(s):  (  11)
```

```
FORMULAS:
  1   [TOP SHAPE] = SQUARE

        Used in rule(s):


  2   [LEG SHAPE] = CIRCULAR

        Used in rule(s):


  3   LENGTH OF EACH SIDE

        Used in rule(s):


  4   [TOP SHAPE] = SQUARE

        Used in rule(s):


  5   [LEG SHAPE] = CIRCULAR

        Used in rule(s):


  6   NUMBER OF NOTCHES

        Used in rule(s):


  7   LENGTH OF EACH SIDE EXISTS

        Used in rule(s):


  8   TOP SHAPE IS NOT SQUARE
```

```
9    [LENGTH EACHSIDE] = YES

      Used in rule(s):


10   [LENGTH EACHSIDE] = YES

      Used in rule(s):


11   [LENGTH1] <= 6

      Used in rule(s):


12   [LENGTH EACHLEG] <= 3

      Used in rule(s):


13   [LEG LENGTH] <= 3

      Used in rule(s):


14   [TOP SHAPE] = SQUARE

      Used in rule(s):


15   [LENGTH EACHSIDE] <= 3

      Used in rule(s):


16   [NOTCHES] <= 6

      Used in rule(s):
```

```
17    [LENGTH EACHSIDE] <= 7

       Used in rule(s):



18    [NUMBER OF LEGS] <= 6

       Used in rule(s):



19    [NUMBER OF LEGS] = [NOTCHES]

       Used in rule(s):



20    3

       Used in rule(s):



       21    [NOTCHES] <= 6

              Used in rule(s):



       22    [NUMBER OF LEGS] <= 6

              Used in rule(s):



       23    [NOTCHES] <= 6

              Used in rule(s):



       24    [NUMBER OF LEGS] <= 6

              Used in rule(s):      3
```

```
25    [NOTCHES] = [NUMBER OF LEGS]

       Used in rule(s):



26    [NOTCHES] = [NUMBER OF LEGS]

       Used in rule(s):



27    [NOTCHES] = [NUMBER OF LEGS]

       Used in rule(s):



28    [COUNTER] + 1

       Used in rule(s):



29    [COUNTER] + 1

       Used in rule(s):



30    [COUNTER] + 1

       Used in rule(s):



31    [COUNTER] = 4

       Used in rule(s):



32    [TOP SHAPE]

       Used in rule(s):



33    [LENGTH EACHSIDE]
```

```
     Used in rule(s):


34   [NOTCHES]

     Used in rule(s):


35   [NUMBER OF LEGS]

     Used in rule(s):


36   [TOP SHAPE] = SQUARE

     Used in rule(s):


37   [TOP SHAPE]

     Used in rule(s):


38   [TOP SHAPE] = "SQUARE"

     Used in rule(s):      4


39   [TOP SHAPE]

     Used in rule(s):


40   [TOP SHAPE]

     Used in rule(s):  (    1)


41   [TOP SHAPE]

     Used in rule(s):  (    5)
```

```
42    [DIAMETER] <= 6

       Used in rule(s):


43    [NOTCHES] = [NUMBER OF LEGS]

       Used in rule(s):       4


44    [NUMBER OF LEGS] <= 6

       Used in rule(s):       7


45    [TOP SHAPE] = "CIRCULAR"

       Used in rule(s):       8


46    [NOTCHES] = [NUMBER OF LEGS]

       Used in rule(s):       8


47    [NOTCHES]

       Used in rule(s):


48    [LENGTH1]

       Used in rule(s):  (    2)


49    [DIAMETER]

       Used in rule(s):  (    6)
```

```
50    [TOP SHAPE] <> "SQUARE"

       Used in rule(s):       9


51    [TOP SHAPE] <> "CIRCULAR"

       Used in rule(s):


52    [TOP SHAPE] <> "CIRCULAR"

       Used in rule(s):      10


53    [NOTCHES] <> [NUMBER OF LEGS]

       Used in rule(s):      11
```

Appendix3

```
/*

  This program is a simulation of the process of space mechanical parts design
  ****************************************************************************


  We simulate the mechanical parts design by designing a table.

  We simulate the guidelines and rules as the design specifiaction of a table.



-----------------------------------------------------------------------------

The design specification of a table is as follows:

     1: top shape ( round or square )
     2: diameter of a round table ( 20 -- 40 )
     3: short side of a rectangular table ( 10 -- 30 )
     4: long side of a rectangular table ( 20 -- 50 )
     5: thickness ( 1 -- 5 )
     6: table top should have notches for leg orientation
     7: leg number equals to the notch number
     8: number of legs ( 4 )
     9: all legs have the same size and the same shape
        to minimize the parts variation
       10: leg shape ( round or square )
       11: leg length ( 25 -- 30 )
       12: legs are all symmetrical

-----------------------------------------------------------------------------

-----------------------------------------------------------------------------
Algorithm:

     step 1: Input the specification range of a table by system.

     step 2: User starts design by choosing:
                1. design by himself
             or 2. design by the system

     step 3: Display the designed table.
             If the user is not satisfied with this design he can choose
                1. modify his design by himself
             or 2. modify the design by the system
             if the user is satisfied with this design go to step 4.

     step 4: Check the design by system to see if there are any violations.
```

```
        Here, the user can choose either
           1. check his design against the standard database
        or 2. check his design against the expanded database

    step 5: If there are no violations go to step 8.

    step 6: If there are any violations
        list the violations as well as the preferred design.
        Here, the user can choose
           1. modify the design by himself ( go to step 3 )
        or 2. modify the design by the system ( go to step 3 )
        or 3. override the suggestions and finish his design
              ( go to step 8)
        or 4. expand the existing database

    step 7: go to step 3

    step 8: Finish the design.
        Here, the user can choose
              1. finish his design and exit the system
        or 2. finish his design and start another design
```

This program is written in an interactive mode giving advice during the design process.

The frame representation for the "table" can also be used by some othe object design which has the similar components like a bench.

This program is composed by 9 modules:

1. main: control all the other functions.
2. initial: set the variables of the extanded database the same as standard database.
3. clear: clear the screen.
4. inputdata: accept data from keyboard and turn them into the appropriate format that the program can accept.
5. sysdesign: design by system with the data from the standard database.
6. display: display the designed table
7. check: check the design against the rules and guidelines in either standard database or extanded database.
8. update: allow the user add more rules in the existing database.
9. redesign: the system modify the user's design so as to meet the requirements of the standard database.

This program is written with "C" in a concise way.

```
*/



#include <stdio.h>
#include <string.h>

#define MAXLEGS 10

char x;

/* x is used for accepting the return key */

int w1=0, w2=0, w3=0, w4=0, w5=0, w6=0, w7=0, w8=0, w9=0, w10=0, w11=0, w12=0;

/*
  w1 to w12 are used for violation1 to violation 12.
  wi=0 means no violation, wi=1 indicates that there is a violation.
*/

float r0=0, r1=1, r2=5, r3=10, r4=20, r5=25, r6=30, r7=40, r8=50,
      rm0, rm1, rm2, rm3, rm4, rm5, rm6, rm7, rm8;
int   r9=4, rm9;
char  rm10, rm11, rm12;


/* ri are for the legal design specification stored in datndard database*/
/*  rmi are for the modified specification stored in expanded database */
/*
   r0 is set to 0.
   If the sides of the table equals r0 that means the shape of
   the top is neither square nor rectangular.
   r1 stands for the lower bound of thickness;
   r2 stands for the upper bound of thickness;
   r3 stands for the lower bound of the short side of the top;
   r4 stands for the lower bound of the long side of the top
      and the lower bound of the diameter;
   r5 stands for the lower bound of the leg length;
   r6 stands for the upper bound of the leg length;
   r7 stands for the upper bound of the diameter;
   r8 stands for the upper bound of the long side of the top;
*/


int   i,j;
/* i is the leg number */
/* j is violation number.
   j=0, there is no violation;
   j>0, there are some violation;
*/

char topshape[12], legshape[12];
```

```
typedef struct
{
 char  top_shape[12], top_notch;
 float diameter, side1, side2, thickness;
 int   notch_num;
} tabletop;

typedef struct
{
 char  leg_shape[12], leg_sym;
 float leg_length;
} tableleg;

struct table
{
  tabletop top;
  tableleg leg[MAXLEGS];
  int      leg_num;
  char     leg_variation;
} t;
/*
  t is the table to be designed.
*/


main()
{

/* This main function decides all the flow control
   implementing the programmer's algorithm.
*/

char c, c0, c1, c2 ;

initial();
/*
  step 1
*/

loop0: initial1();
       c='0';
/*
   The user is going to design a table other than the first one
   therefore it is necesary to reset the initial value.
*/

while ( c!='1' && c!='2')
/*
  step 2
*/
  {
```

```
  printf("\n\n\n");
  printf("1 -- design the table by yourself\n");
  printf("2 -- design the table by system\n");
  printf("Please enter your choice\n");
  scanf("%c%c",&c,&x);
  clear();
  if (c=='1')
     inputdata();
  if (c=='2')
     sysdesign();
  }

loop1:
      c0='n';
while (c0!='y')
  {
/*
   step 3
*/
  clear();
  displaydata();
  printf("\n\n\n");
  printf("Are you satisfied with this design specification ? \n");

  loop2:
  printf(" 1 -- satisfied and checking the rules and the guidelines\n");
  printf(" 2 -- modify the design by yourself\n");
  printf(" 3 -- modify the design by the system\n");
  printf(" Please enter your choice\n");
  scanf("%c%c",&c0,&x);
  clear();
  if (c0=='1')
     goto chk;
     else
     if (c0=='2')
     {
      inputdata();
      goto loop1;
     }
     else
     if (c0=='3')
        {
         sysdesign();
         goto loop1;
        }
        else
        goto loop2;
  }

chk:
check();
/*
```

```
  step 4
*/
if ( j!=0 )
   {
   printf("\n\nAre you going to override these suggestions ? \n");
   printf("1 -- modify your design by yourself\n");
   printf("2 -- modify your design by the system\n");
   printf("3 -- override these suggestions and finish design\n");
   printf("4 -- expanding the existing database\n");
   printf("Please enter your choice\n");
   printf("Hit '0' to finish your design\n");
   scanf("%c%c",&c1,&x);
   clear();
   if (c1=='1')
     {
     inputdata();
     goto loop1;
     }
   if (c1=='2')
     {
     redesign();
     goto loop1;
     }
   if (c1=='4')
     {
     update();
     goto loop1;
     }
     else
     goto end;
   }
   else
   {
   end:
/*
   step 8
*/
       printf("1 -- finish the design and exit the design system\n");
       printf("2 -- design another table\n");
       scanf("%c%c",&c2,&x);
       clear();
       if (c2=='2')
       goto loop0;
       else
       {
        printf(" Congretulation!\n\n");
        printf(" You have finished designing the table.\n\n");
        displaydata();
        printf("\n\n\n\n Hit  'ENTER'  key to exit.\n\n\n");
        scanf("%c",&x);
        clear();
        }
   }
}


initial()

/*
  Set the initial value of the database.
  Let the expanded database the same as the original database.
*/

{
 rm0=r0;
 rm1=r1;
 rm2=r2;
 rm3=r3;
 rm4=r4;
 rm5=r5;
 rm6=r6;
 rm7=r7;
 rm8=r8;
 rm9=r9;
 rm10=rm11=rm12='y';
 printf("You are entering the computer designing system\n");
 printf("Your are now designing a table\n");
 printf("There are 12 rules stored in the database for the table designing\n");
 printf("They are as follows:\n");
 printf("1: top shape ( round or square )\n");
 printf("2: diameter of a round table ( 20 -- 40 )\n");
 printf("3: short side of a rectangular table ( 10 -- 30 )\n");
 printf("4: long side of a rectangular table ( 20 -- 50 )\n");
 printf("5: thickness ( 1 -- 5 )\n");
 printf("6: table top should have notches for leg orientation\n");
 printf("7: leg number equals to the notch number\n");
 printf("8: number of legs ( 4 )\n");
 printf("9: all legs have the same size and the same shape\n");
 printf("   to minimize the parts variation\n");
 printf("10: leg shape ( round or square )\n");
 printf("11: leg length ( 25 -- 30 )\n");
 printf("12: legs are all symmetrical\n");
}


initial1()

/*
   This function is to set the initial value for the table to be designed.
   It is especially necesary when the user wants to design a table other
   than his first one.
*/
```

```
{
 int k;

/* k is the number of legs */

 strcpy(t.top.top_shape,"  ");
 t.top.top_notch=NULL;
 t.top.diameter=0;
 t.top.side1=0;
 t.top.side2=0;
 t.top.thickness=0;
 t.top.notch_num=0;
 for (k=1; k<MAXLEGS; k++)
     {
     strcpy(t.leg[k].leg_shape,"  ");
     t.leg[k].leg_sym=NULL;
     t.leg[k].leg_length=0;
     }
 t.leg_num=0;
 t.leg_variation=NULL;
}


clear()

/*
  This function is to clear the screen.
*/

{
printf("\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n");
}



inputdata()

/*
  This function is to create an environment for the user to input
  the parameters of the table being designed.
*/

{
char c1, c2;

i=1;

printf("        Now let's start  input or modify the data of the table.\n\n");
printf("************************************************************");
printf("*********\n");
printf("Do you want to input or modify the table top?(y/n)\n");
scanf("%c%c",&c1,&x);
```

```
if (c1=='y')
   {
    printf("Do you want to input or modify  the shape of the table top?");
    printf("(y/n)\n");
    scanf("%c%c",&c1,&x);
    if (c1=='y')
     {
     printf("What kind of shape do you like for the table?\n");
     printf("(The suggested shape is either round or square ");
     printf("for the assembly convenience)\n");
     scanf("%s%c",t.top.top_shape,&x);
     }
     if (strcmp(t.top.top_shape,"round")==0) c2='a';
     if (strcmp(t.top.top_shape,"square")==0) c2='b';
     if (strcmp(t.top.top_shape,"rectangular")==0) c2='c';
    switch(c2)
     {
     case'a':
          printf("Do you want to input or modify the diameter ");
          printf("of the round table (y/n)?\n");
          scanf("%c%c",&c1,&x);
          if (c1=='y')
               {
               printf("Please input the diameter of the table.\n");
               printf("(The suggested diameter is 20 -- 40)\n");
               scanf("%f%c",&t.top.diameter,&x);
               }
          break;
     case'b':
          printf("Do you want to input or modify the size ");
          printf("of the table side?(y/n)\n");
          scanf("%c%c",&c1,&x);
          if (c1=='y')
               {
               printf("Please input the size of the table side.\n");
               printf("(The suggested table side is 10 -- 50 )\n");
               scanf("%f%c",&t.top.side1,&x);
               t.top.side2=t.top.side1;
               }
          break;
     case'c':
          printf("Do you want to input or modify the size ");
          printf("of the table sides?(y/n)\n");
          scanf("%c%c",&c1,&x);
          if (c1=='y')
               {
               printf("Please input the size of short side.\n");
               printf("(The suggested short side is 10 -- 30)\n");
               scanf("%f%c",&t.top.side1,&x);
               printf("Please input the size of long side.\n");
               printf("(The suggested long side is 20 -- 50)\n");
               scanf("%f%c",&t.top.side2,&x);
```

```
                }
           break;
      default: break;
      }
    printf("Do you want to input or modify the thickness ");
    printf("of the table?(y/n)\n");
    scanf("%c%c",&c1,&x);
    if (c1=='y')
     {
     printf("Please input the thickness of the table. \n");
     printf("(The suggested thickness of the table top is 1 -- 5 )\n");
     scanf("%f%c",&t.top.thickness,&x);
     }
    printf("Is there any notch on the top ?(y/n)\n");
    printf("(The notches on the table top is for the convenience ");
    printf("of leg orientation)\n");
    scanf("%c%c",&t.top.top_notch,&x);
      if (t.top.top_notch=='y')
         {
          printf("Please input the number of notches.\n");
          printf("(For the convenience of leg orientation\n");
          printf("it's better to have the number of notches ");
          printf("the same as the number of legs.)\n");
          scanf("%d%c",&t.top.notch_num,&x);
         }
         else
         {
         t.top.top_notch='n';
         t.top.notch_num=0;
         }
   }
printf("Do you want to input or modify the table legs?(y/n)\n");
scanf("%c%c",&c1,&x);
if (c1=='y')
   {
    printf("Do you want to input or modify the number of legs?(y/n)\n");
    scanf("%c%c",&c1,&x);
    if (c1=='y')
     {
     printf("Please input the number of table legs.\n");
     printf("(The suggested number of legs is 4)\n");
     scanf("%d%c",&t.leg_num,&x);
     }
    printf("Are all the legs have the same size and the same shape ?(y/n)\n");
    printf("(Same legs can minimize part variation.)\n");
    scanf("%c%c",&c1,&x);
    if (c1=='y')
       t.leg_variation='n';
       else
       t.leg_variation='y';
    loop2:
    printf("You are now modifying leg[%d].\n",i);
    printf("Do you want to input or modify the shape of this leg?(y/n)\n");
    scanf("%c%c",&c1,&x);
    if (c1=='y')
     {
     printf("Please input the shape of this leg (round, square,...).\n");
     printf("(The suggested shape of table leg is)");
     printf("either round or square)\n");
     scanf("%s%c",t.leg[i].leg_shape,&x);
     }
    printf("Is this leg symmetrical in shape?(y/n)\n");
    printf("(symmetrical leg gives great convenience for its feeding)\n");
    scanf("%c%c",&t.leg[i].leg_sym,&x);
    printf("Do you want to input or modify the length of this leg?(y/n)\n");
    scanf("%c%c",&c1,&x);
    if (c1=='y')
     {
     printf("Please input the length of this leg.\n");
     printf("(The suggested leg length is 25 -- 30)\n");
     scanf("%f%c",&t.leg[i].leg_length,&x);
     }
    if (t.leg_variation=='n')
       for (i=2; i<t.leg_num+1; i++)
        t.leg[i]=t.leg[1];
       else
        {
        i++;
        if (i<t.leg_num+1)
        goto loop2;
        }
   }
printf("\n\n\n");
}


sysdesign()

/*
  This function is used to design the table by the system
  e.i. the system decide all the parameters of the table.
*/

{
int k;

 strcpy(t.top.top_shape,"round");
 t.top.diameter=30;
 t.top.side1=0;
 t.top.side2=0;
 t.top.thickness=3;
 t.top.top_notch='y';
 t.top.notch_num=4;
```

```
  for (k=1; k<5; k++)
      {
      strcpy(t.leg[k].leg_shape,"round");
      t.leg[k].leg_sym='y';
      t.leg[k].leg_length=28;
      }
  t.leg_num=4;
  t.leg_variation='n';
 }


displaydata()

/*
  This function is to display the parameters of the designed table.
*/

{
char c;
int i1;

i=1;
printf("The designed table is as follows\n");
printf("------------------------------------------------------------\n");
printf("The topshape is %s\n",t.top.top_shape);
     if (strcmp(t.top.top_shape,"round")==0) c='a';
     if (strcmp(t.top.top_shape,"square")==0) c='b';
     if (strcmp(t.top.top_shape,"rectangular")==0) c='c';
switch(c)
     {
     case 'a': printf("The diameter of the table is %f\n",t.top.diameter);
            break;
     case 'b': printf("The side of the table is %f\n",t.top.side1);
            break;
     case 'c':
          printf("The sides of the table are %f",t.top.side1);
          printf("  by  %f\n",t.top.side2);
          break;
     default:break;
     }
printf("The thickness of the table is %f\n",t.top.thickness);
if (t.top.top_notch=='y'|| t.top.notch_num!=0)
   printf("There are %d top notches\n",t.top.notch_num);
   else
   printf("There are no top notches for the leg orientation!\n");
printf("The number of legs is %d\n",t.leg_num);
if (t.leg_variation=='n')
   i1=1;
   else
   i1=t.leg_num;
for (i=1; i<i1+1; i++)
```

```
    {
     printf("The shape of the leg[%d] is %s\n",i,t.leg[i].leg_shape);
     if (t.leg[i].leg_sym=='y')
     printf("leg[%d] is symmetrical\n", i);
     else
     printf("leg[%d] is NOT symmetrical!\n",i);
     printf("The length of leg[%d] is %f\n",i,t.leg[i].leg_length);
    }
if (t.leg_variation=='n')
   printf("All legs have the same size and the same shape\n");
printf("--------------------------------------------------------\n");
}



check()

/*
  This function is to check the design against the database.
  c0=1, check against the standard database;
  c0=2, check against the extended database;
*/

{
char c0,c1,c2;

j=0;
loopcheck:
printf("1 -- Check against the standard database\n");
printf("2 -- Check against the expanded database\n");
printf("     Please enter your choice\n");
scanf("%c%c",&c0,&x);
clear();
if ((c0!='1') && (c0!='2'))
   goto loopcheck;
if (strcmp(t.top.top_shape,"round")==0) c1='a';
if (strcmp(t.top.top_shape,"square")==0) c1='a';
switch(c1)
  {
  case'a': ; break;
  default:
    if (c0=='1')
       {
     j++;
     w1=1;
     printf("violation %d :      ",j);
     printf("table topshape is neither round nor square\n");
     printf("Your design is %s.             \n",t.top.top_shape);
     printf("The preferred topshape is either round or square.\n");
     }
    if (c0=='2')
     {
          if (strcmp(t.top.top_shape,topshape)==0)
             break;
             else
             {
           j++;
           w1=1;
           printf("violation %d :      ",j);
           printf("table topshape cannot be found in database\n");
           printf("Your design is %s.             \n",t.top.top_shape);
           printf("The preferred topshape is either round or square.");
           printf("\n\n");
             }
      }
    break;
  }
if (strcmp(t.top.top_shape,"round")==0)
   {
   if ((c0=='1') && (t.top.diameter<r4 || t.top.diameter>r7))
     {
     j++;
     w2=1;
     printf("violation %d :      ",j);
     printf("the diameter is out of range.\n");
     printf("Your design is %f.    ",t.top.diameter);
     printf("The preferred diameter is between 20 to 40.\n");
     }
   if ((c0=='2') && (t.top.diameter<rm4 || t.top.diameter>rm7))
     {
     j++;
     w2=1;
     printf("violation %d :      ",j);
     printf("the diameter is out of range.\n");
     printf("Your design is %f.    ",t.top.diameter);
     printf("The preferred diameter is between 20 to 40.\n");
     }
   }
if (t.top.side1!=r0)
   {
   if ((c0=='1') && (t.top.side1<r3 || t.top.side1>r6))
     {
     j++;
     w3=1;
     printf("violatio %d :                  ",j);
     printf("side1 is out of range.\n");
     printf("Your design is %f.      ",t.top.side1);
     printf("The preferred side1 is between 10 to 30.\n");
     }
   if ((c0=='2') && (t.top.side1<rm3 || t.top.side1>rm6))
     {
     j++;
     w3=1;
     printf("violatio %d :                  side1 is out of range.\n",j);
```

```
       printf("Your design is %f.      ",t.top.side1);
       printf("The preferred side1 is between 10 to 30.\n");
       }
     }
  if (t.top.side2!=r0)
     {
     if ((c0=='1') &&  (t.top.side2<r4  ||  t.top.side2>r8))
       {
       j++;
       w4=1;
       printf("violatio %d :                 side2 is out of range.\n",j);
       printf("Your design is %f.    ",t.top.side2);
       printf("The preferred side2 is between 20 to 50.\n");
       }
     if ((c0=='2') &&  (t.top.side2<rm4  ||  t.top.side2>rm8))
       {
       j++;
       w4=1;
       printf("violatio %d :                 side2 is out of range.\n",j);
       printf("Your design is %f.    ",t.top.side2);
       printf("The preferred side2 is between 20 to 50.\n");
       }
     }
  if ((c0=='1') && (t.top.thickness<r1  ||  t.top.thickness>r2))
     {
     j++;
     w5=1;
     printf("violation %d :                thickness is out of range.\n",j);
     printf("Your design is %f.     ",t.top.thickness);
     printf("The preferred thickness is between 1 to 5.\n");
     }
  if ((c0=='2') && (t.top.thickness<rm1  ||  t.top.thickness>rm2))
     {
     j++;
     w5=1;
     printf("violation %d :                thickness is out of range.\n",j);
     printf("Your design is %f.     ",t.top.thickness);
     printf("The preferred thickness is between 1 to 5.\n");
     }
  if ((c0=='1') && (t.top.top_notch!='y'))
     {
     j++;
     w6=1;
     printf("voilation %d :      no notches for the leg orientation!\n",j);
     printf("The preferred design is to have notches on the top ");
     printf("for leg orientation\n");
     }
  if ((c0=='2') && ((t.top.top_notch!='y') && (t.top.top_notch!=rm10)))
     {
     printf("voilation %d :      no notches for the leg orientation!\n",j);
     printf("The preferred design is to have notches on the top ");
     printf("for leg orientation\n");
```

```
   }
 if ((c0=='1') && (t.top.notch_num!=t.leg_num))
   {
   j++;
   w7=1;
   printf("violation %d :      notch# is not equal to leg#!\n",j);
   printf("notch number is %d, leg number is %d\n",t.top.notch_num,t.leg_num);
   printf("Please modify the notch number\n");
   }
 if ((c0=='1') && (t.leg_num!=r9))
   {
   j++;
   w8=1;
   printf("violation %d :                legs are not 4!\n",j);
   printf("Your design is %d.            ",t.leg_num);
   printf("The preferred design is 4.\n\n");
   }
 if ((c0=='2') && (t.leg_num!=rm9))
   {
   j++;
   w8=1;
   printf("violation %d :      leg number cannot be found in database!\n",j);
   printf("Your design is %d.            ",t.leg_num);
   printf("The preferred design is 4.\n\n");
   }
 if ((c0=='1') && (t.leg_variation!='n'))
   {
   j++;
   w9=1;
   printf("voilation %d :                    legs are different\n",j);
   printf("your design  causes assembly difficulties\n");
   printf("The preferred design is to minimize parts variation\n");
   }
 if ((c0=='2') && ((t.leg_variation!='n') && (t.leg_variation!=rm11)))
   {
   j++;
   w9=1;
   printf("voilation %d :                    legs are different\n",j);
   printf("your design  causes assembly difficulties\n");
   printf("The preferred design is to minimize parts variation\n");
   }
 for (i=1; i<t.leg_num+1; i++)
    {
    if ((c0=='1') && (t.leg[i].leg_sym!='y'))
       {
       j++;
       w10=1;
       printf("violation %d :                 leg is not symmetrical.\n",j);
       printf("your design for leg[%d] causes assembly difficulties.\n",i);
       printf("The preferred design is to have all legs symmetrical.\n");
       }
    if ((c0=='2') && ((t.leg[i].leg_sym!='y') && (t.leg[i].leg_sym!=rm12)))
```

```
       {
       j++;
       w10=1;
       printf("violation %d :                leg is not symmetrical.\n",j);
       printf("your design for leg[%d] causes assembly difficulties.\n",i);
       printf("The preferred design is to have all legs symmetrical.\n");
       }
    }
for (i=1; i<t.leg_num+1; i++)
    {
    if ((strcmp(t.leg[i].leg_shape,"round")==0)) c2='a';
    if ((strcmp(t.leg[i].leg_shape,"square")==0)) c2='a';
    switch(c2)
      {
      case'a': ; break;
      default:
     if (c0=='1')
          {
          j++;
          w11=1;
          printf("violation %d :     ",j);
          printf("legshape is neither round nor square.\n");
          printf("Your design for leg[%d]          ",i);
          printf(" is %s.        \n",t.leg[i].leg_shape);
          printf("The preferred legshape is either round or square.\n");
          }
     if (c0=='2')
        {
         if (strcmp(t.leg[i].leg_shape,legshape)==0)
            break;
            else
            {
          j++;
          w11=1;
          printf("violation %d :     ",j);
          printf("legshape cannot be found in database.\n");
          printf("Your design for leg[%d]",i);
          printf(" is %s.            \n",t.leg[i].leg_shape);
          printf("The preferred legshape is either round or square.");
          printf("\n");
          }
        }
     break;
      }
  }
for (i=1; i<t.leg_num+1; i++)
    {
    if ((c0=='1') && (t.leg[i].leg_length<r5  ||  t.leg[i].leg_length>r6))
     {
     j++;
     w12=1;
     printf("violation %d :      leglength is out of range.\n",j);
     printf("Your design for leg[%d] is %f. ",i,t.leg[i].leg_length);
     printf("The preferred design is between 25 to 30.\n");
     }
    if ((c0=='2') && (t.leg[i].leg_length<rm5  ||  t.leg[i].leg_length>rm6))
     {
     j++;
     w12=1;
     printf("violation %d :      leglength is out of range.\n",j);
     printf("Your design for leg[%d] is %f. ",i,t.leg[i].leg_length);
     printf("The preferred design is between 25 to 30.\n");
     }
    }
if ((j==0) && (c0=='1'))
   {
   printf("No violation against ");
   printf("the rules and guidelines in standard database\n\n\n\n\n\n");
   }
if ((j==0) && (c0=='2'))
   {
   printf("No violation against the ");
   printf("rules and guidelines in the extanded database\n\n\n\n\n\n");
   }
}


update()

/*
  This function allows the user expand the standard database.
*/

{
int  m;
char ch;

m=0;
while (m==0)
  {
  clear();
  printf("    Standard database                        Extended database\n");
  printf("--------------------------------------------------------------\n");
  if (w1==1)
  printf("a: topshape ( round or square )             %s\n",topshape);
  if (w2==1)
  printf("b: diameter ( 20 -- 40 )                    %f -- %f\n",rm4,rm7);
  if (w3==1)
  printf("c: side1 ( 10 -- 30 )                       %f -- %f\n",rm3,rm6);
  if (w4==1)
  printf("d: side2 ( 20 -- 50 )                       %f -- %f\n",rm4,rm8);
  if (w5==1)
  printf("e: thickness ( 1 -- 5 )                     %f -- %f\n",rm1,rm2);
  if (w6==1)
  printf("f: table top has notches                          %c\n",rm10);
  if (w7==1)
  printf("g: leg# = notch#                                  leg# <> notch#\n");
  if (w8==1)
  printf("h: number of legs ( 4 )                           %d\n",rm9);
  if (w9==1)
  printf("i: all legs are the same                          %c\n",rm11);
  if (w10==1)
  printf("j: each leg is symmetrical                        %c\n",rm12);
  if (w11==1)
  printf("k: legshape (round or square)                     %s\n",legshape);
  if (w12==1)
  printf("l: leg length ( 25 -- 30 )                        %f -- %f\n\n\n",rm5,rm6);
  printf("Please  input  the  choice  which  you  want  to  modify  the  above
specification\n");
  printf("Hit '0'  key  to exit modify the database.\n");
  scanf("%c%c",&ch,&x);
  switch(ch)
       {
       case 'a':
         printf("Please input the topshape of the table\n");
         scanf("%s%c",topshape,&x);
         w1=0;
         break;
       case 'b':
         printf("Please input the lower bound of the diameter.\n");
         scanf("%f%c",&rm4,&x);
         printf("Please input the upper bound of the diameter.\n");
         scanf("%f%c",&rm7,&x);
         w2=0;
         break;
       case 'c':
         printf("Please input the lower bound of side1.\n");
         scanf("%f%c",&rm3,&x);
         printf("Please input the upper bound of side1.\n");
         scanf("%f%c",&rm6,&x);
         w3=0;
         break;
       case 'd':
         printf("Please input the lower bound of side2.\n");
         scanf("%f%c",&rm4,&x);
         printf("Please input the upper bound of side2.\n");
         scanf("%f%c",&rm8,&x);
         w4=0;
         break;
       case 'e':
         printf("Please input the lower bound of thickness.\n");
         scanf("%f%c",&rm1,&x);
         printf("Please input the upper bound of thickness.\n");
         scanf("%f%c",&rm2,&x);
         w5=0;
         break;
       case 'f':
         printf("Please enter 'n', if you don't want ");
         printf("to have notch on the table top\n");
         scanf("%c%c",&rm10,&x);
         w6=0;
         break;
       case 'g':
         w7=0;
         break;
       case 'h':
         printf("Please input the number of legs.\n");
         scanf("%d%c",&rm9,&x);
         w8=0;
         break;
       case 'i':
         printf("Please enter 'n', if you don't want all legs the same\n");
         scanf("%c%c",&rm11,&x);
         w9=0;
         break;
       case 'j':
         printf("Please enter 'n', if you don't want ");
         printf("the leg to be symmetrical\n");
         scanf("%c%c",&rm12,&x);
         w10=0;
         break;
       case 'k':
         printf("Please input the legshape of the table\n");
         scanf("%s%c",topshape,&x);
         w11=0;
         break;
       case 'l':
         printf("Please input the lower bound of leg length.\n");
         scanf("%f%c",&rm5,&x);
         printf("Please input the upper bound of leg length.\n");
         scanf("%f%c",&rm6,&x);
         w12=0;
         break;
       default:
         m=1;
         break;
       }
     }
}


redesign()

/*
   This function helps the user modify his design.
   If the use's design has some violations against the
   rules and the guidelines, this module can eliminate
   these violations automatically.
*/

{
int l;

if (w1==1)
     {
     strcpy(t.top.top_shape,"round");
     t.top.diameter=30;
     t.top.side1=0; t.top.side2=0;
     w1=0;
     }
if (w2==1)
     {
     if ( t.top.diameter<20 ) t.top.diameter=20;
          else t.top.diameter=40;
     w2=0;
     }
if (w3==1)
     {
     if ( t.top.side1<10 ) t.top.side1=10;
          else t.top.side1=30;
     w3=0;
     }
if (w4==1)
     {
     if (t.top.side2<20 ) t.top.side2=20;
          else t.top.side2=50;
     w4=0;
     }
if (w5==1)
     {
     if ( t.top.thickness<1 ) t.top.thickness=1;
          else t.top.thickness=5;
     w5=0;
     }
if (w6==1)
     {
     t.top.top_notch='y';
     t.top.top_notch=4;
     w6=0;
     }
if (w7==1)
     {
     t.top.notch_num=t.leg_num;
     w7=0;
     }
if (w8==1)
     {
     t.leg_num=4;
     w8=0;
     }
if (w9==1)
     {
     t.leg_variation='n';
     w9=0;
     }
if (w10==1)
     {
     for (l=1; l<5; l++)
         t.leg[l].leg_sym='y';
     w10=0;
     }

if (w11==1)
     {
     for (l=1; l<5; l++)
         strcpy(t.leg[l].leg_shape,"round");
     w11=0;
     }
if (w12==1)
     {
     if ( t.leg[1].leg_length<25 )
        for (l=1; l<5; l++)
            t.leg[l].leg_length=25;
        else
        for (l=1; l<5; l++)
            t.leg[l].leg_length=30;
     w12=0;
     }
}
```